International Journal of Information and Computer Science
IJICS Volume 1, Issue 3, June 2012 PP. 64-72
ISSN(online) 2161-5381
ISSN(print) 2161-6450
www.iji-cs.org

Raising User Acceptance of Token-based Authentication by Single Sign-On

Sandro Wefel and Paul Molitor
Institute for Computer Science, Martin-Luther-University Halle-Wittenberg, Halle (Saale), Germany
Email: sandro.wefel@informatik.uni-halle.de, paul.molitor@informatik.uni-halle.de

(Abstract) Secure and reliable authentication is one important part in the scope of IT security, but the practical experience shows that adequate authentication mechanisms are often inconvenient for the populace. Password-based mechanisms dominate as they are easy to use and do not need additional hardware. However, lots of users do not realize the threads with respect to password-based mechanisms: many users use passwords for remote authentication over unsecured channels or in not trustworthy environments, or repeatedly use the same password. Mechanisms based on asymmetric cryptographic operations in combination with hardware token may solve these security problems but are less comfortable in usage. In order to raise user acceptance of these methods we combine them with client-based single sign-on for local and remote authentication. This combined approach leads to a win-win-situation: token-based authentication reduces the risk of stolen authentication factors as the tokens are protected against misuse by a secret pin and it does not need much more user effort than password-based mechanisms. Furthermore, the approach can be used for further IT security applications, e.g., electronic signatures or encipherment, which raises user acceptance even further.

Keywords: authentication; single sign-on; crypto token

1. INTRODUCTION

Life in the beginning of the 21st century is strongly governed by the application of connected digital technologies. Services as interactive television, just-in-time access on worldwide distributed information, online libraries, online publishing or online banking are only a few examples of our new connected society. This mainstream trend is going on. However, security problems related to the basic technologies can bring people not to apply such network services.

In addition to the establishment of secure electronic communication channels and malpractice of personal data by administration staff, effective user authentication, which is also responsible for the assignment of the privileges to the user who is logged in, is most important to overcome security problems. An attacker which is wrongfully authenticated as the administrator of the system has access to the whole system; an attacker which authenticates with a stolen user ID has access to the private data of that person.

There are several different methods for authenticating. Each of them requires that an *authentication factor* of the *supplicant*, which is an individual information or an object, is presented to the *authentication server* –authentication factors are classified as either *knowledge factor* (authentication server and supplicant share a secret, e.g., a password or a personal identification number) or *ownership factor* (the supplicant owns an object, which cannot be copied or which can only be hardly duplicated), or *inherence factor* (a factor which is permanently connected to the supplicant, e.g., a biometric criterion) or derived factors [1] [2]. The authentication factor has to fulfill some quality criteria, e.g., with respect to the chosen password in the case of username/password-authentication, and has to be protected against unauthorized access. These requirements are problematic particularly with regard to a broad community of users with very different technical expert knowledge [3]. In particular, users underestimate the possible secondary consequences of circulating her authentication factors. On the part of the users, the need of protecting the authentication factor is not always understood; often authentication factors for logging in an IT system or accessing the WLAN are handed to other persons. More than two third of the users use less than seven distinct passwords for all applications and websites, up to 30% share passwords with other users [3] [4].

These facts demand more comfortable approaches for authentication to computer systems and network services; moreover, the authentication factors should be prevented against duplication or misuse to reduce the human factor [5] [6].

Biometrical methods have the advantage that the authentication factor is always available and cannot be duplicated in general. However, they require elaborated hardware. The problem consists in the evaluation of the non-immutable analog value from the scanner applied to the biometric input. Either the analog input needs to be mapped on a unique digital value to compare it with the respective

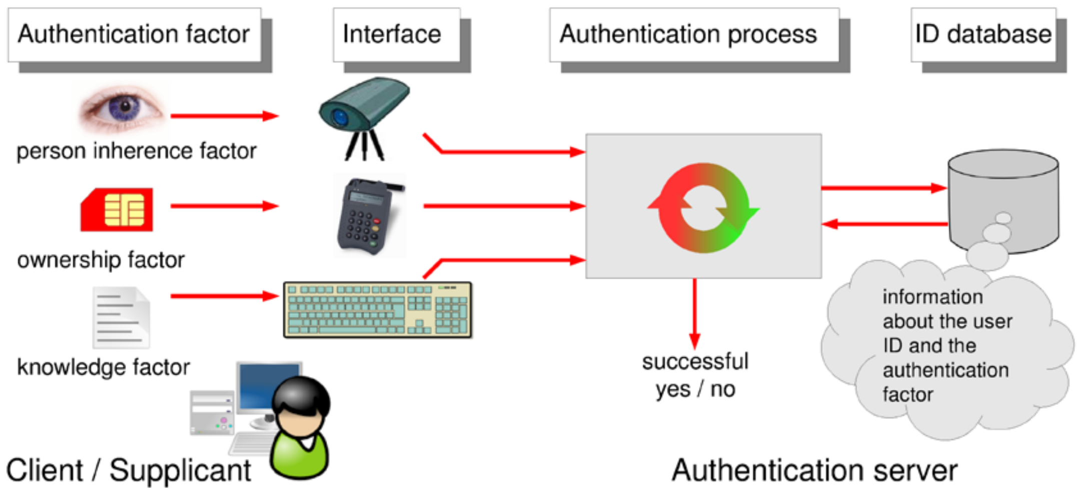

Figure 1. Authentication factors

value stored in the identity (ID) database or the comparator has to work with fuzzy data. Moreover, some users still disapprove of biometry because they mistrust the technology and the safeguarding of the personal biometrical data [7] [8]. For these reasons we do not discuss biometrical methods in the following.

Smart cards or other forms of hardware tokens allow the user to keep information, e.g., private keys and passwords, in safe custody. They are especially suitable for being used in the context of public key systems when provided with interfaces to call methods requiring vital information. Such hardware tokens ensure that the private keys never leave the token under any circumstances and thus enhance reliability to a large extent. For protection against misuse after token loss, the user has to authenticate herself to the token. She shows to be legitimate by sending secret information to the token, e.g., a pin – this process of authentication to the hardware token is called *Card-Holder-Verification* (CHV). After authentication of the user to the token, the token offers methods to read protected information or to use signature or other cryptographic operations that can be used for different purposes. In particular the cryptographic operations can be used for authentication of the token owner to other systems. These operations can be executed by the processor which is integrated on the token once authentication to the hardware token has succeeded. Accordingly, it is ensured that the authentication factors cannot be duplicated. Thus, the misuse of authentication factors is hard when stored in hardware tokens. Moreover, the pin is the only secret information the user has to keep in mind. Thus, it could be chosen by the user such that it is not easy to be guessed. However, that won't solve our case as the pin has to be put in very often during a session, e.g., for each application the user calls. In order to reduce time and effort, most users will still choose a short pin to authenticate to the hardware token. So an attacker who plans to steal the token can easily obtain the corresponding pin by glimpsing at its input. Thus the number of pin inputs should be reduced for enhancing reliability as in this case there is hope that the user chooses a more complex number combination as pin. This leads us to *single sign-on* and thus to a better user acceptance of using hardware tokens.

The paper presents a method for secure hardware token-based user authentication which allows single sign-on for multiple client applications (possibly) distributed over several servers without modifications on server side. Whereas current client-based hardware token approaches store passwords for authenticating the user to the applications, the approach presented here uses the user certificate stored in the token, whereby the pin of the token has to be put in only once and not each time a new application is called. Thus, an increase of the consumer acceptance comes along with more security.

The paper is structured as follows. First of all we review and discuss the different authentication methods and hardware token-based authentication, in particular with respect to certificate-based methods. Next we present our new approach for hardware token-based authentication with single sign-on capability which works on client side and does not need to modify server software (provided that the server allows certificate-based authentication).

2. AUTHENTICATION

Authentication can be divided in two types, local authentication and network authentication. The two authentication types differ in the place, where the claimed user ID is verified. The first covers all authentication mechanism for the login to a local machine or service. For non-local authentication the ID is verified by a service on a remote machine, e.g., a web server; thus the authentication information (user credentials) leaves the local machine scope. Such an approach offers new attack surfaces in comparison to local authentication [9].

In the following subsections we discuss the different authentication methods with respect to security and user's acceptance for both, local and non-local authentication.

2.1. Local Authentication Methods

Computers exist in different forms, e.g., as office computers or in cell phones to mention just two. Most of these systems require an authentication of the user: the office computer requires a user login, a cell phone the input of the pin which is associated to the SIM card. These systems have in common that the process to verify the asserted identity runs in the local machine itself. The user credential, which is derived from the authentication factor, is only transported from the user to the local machine over secure connections, to start from the assumption that the local machine is not infected by harmful software. Apart from the potential weak spots which go along with keyboards or smartcard terminals, no extra efforts to prevent abuse are needed.

Local authentication covers all the authentication schemes and mechanisms where the authentication process takes place in the local secure environment. Figure 2 shows several authentication schemes for local authentication. All these schemes have in common that in a first step the user tells her identity, the so called *user ID*. In a second step the user uses her own authentication factor to prove the asserted ID. The ID and the authentication information, which is derived from the individual authentication factor, are transported from the computer interface to the authentication service on the local machine. The authentication service uses the ID to inquire the combined authentication information from the ID database. The authentication is successful if the database value matches the value from the user.

The most common knowledge-based local authentication mechanism is shown in Fig. 2(a). The authentication factor is a password. As the ID database may be outside the secure machine environment, the information stored in the database has to be protected. A common solution consists of storing a non-reversible value $f_{HASH}(p)$ derived from the password p instead of the password p itself. Given a hash-value h it should be practically impossible to find a value p with $f_{HASH}(p) = h$. However, username/password-authentication is problematic as we have already argued above.

Authentication based on ownership factors are shown in Fig. 2(b)-(d). The ownership in Fig. 2(b) consists of a value K_{user}

Figure 2. Local authentication schemes

from which a time limited authentication information is generated. This information can be used as time-limited password. Because the authentication information is only short-time or one-time usable, bystanders which could catch a glimpse on the keyboard do not pose a danger. However, this approach needs a synchronized time base. Moreover, the customer has to work with extra hardware and there is no added value but secure authentication, which comes with this additional hardware.

One-time passwords (OTP) work like transaction numbers (TANs) in financial transactions (see Fig. 2(c)). The customer gets a list with one-time usable authentication values. After authentication the ID/OTP combination is deleted from the database. The OTP-approach has the same advantages as time-based passwords, but it does not require a synchronized time base. The owner has to cover the OTP list in a safe custody and if the list is exhausted, new OTPs have to be generated and securely interchanged between the authentication service and the user which again is additional overhead.

Figure 2(d) shows another authentication mechanism with one time usable values. The authentication service generates a challenge *cha* and tells it to the user (more precisely, to a proper hardware owned by the user, e.g., a processor smartcard – in the following, we refer to both, user and smartcard as supplicant). Both, supplicant and authentication service calculate the corresponding response $f(cha,K_{user})$ which does not only depend on the challenge *cha* but also on the personal authentication factor K_{user}. The authentication is successful if the supplicant submits the fitting response. If each challenge is only used once, then the response is not abusive reusable. The combination with safe non-recoverable storage of value K_{user} in a user token yields a reliable authentication mechanism. Once again, the customer has to work with extra hardware and has to transport a smartcard or a token.

2.2. Remote Authentication

For *network authentication*, also referred to as *remote client authentication*, the supplicant on the client side is connected to the authenticator over a potentially insecure network, e.g., the internet or a wireless channel between notebooks and access points.

The methods listed in Fig. 2(a)-(d) as well as the asymmetric method (Fig. 3) can be applied for remote authentication, too, and have the same pros and cons as in local authentication [9].

The authentication factor is either processed in the supplicant application for authentication to a network service or is transported over the network to the service. Nondisclosure of the private authentication factor must be ensured, in particular if these confidential values are transported over public networks. This can be done by establishing a secure tunnel for the information transport or by using methods with non-sensitive information transport, e.g., the challenge-response mechanisms with asymmetric key pairs.

However, in general, the challenge for remote authentication is to ensure that the authentication server is the expected server, i.e., a client which sends sensitive information to the server must not be compromised by a man-in-the-middle attack. To prevent man-in-the-middle attacks the server has to authenticate to the client before the reverse client-to-server authentication. The server needs authentication information, which is accepted by the client as trustworthy. Foreign servers, where the client has not obtained any trust information before the first time connection, compound the problem. A common solution to this problem is to use digital certificates in combination with challenge response authentication. The particular advantage of certificate-based authentication is that it is not limited to server-to-client authentication, but can also be applied to client-server authentication.

We review both, authentication based on asymmetric cryptography and certificate-based authentication in the following.

Authentication based on Asymmetric Cryptography

The authentication methods listed above require that some information is safely stored in the database to prevent against misuse. In particular, value K_{user} of the method shown in Fig. 2(d) should not be recoverable from the challenge *cha* and the

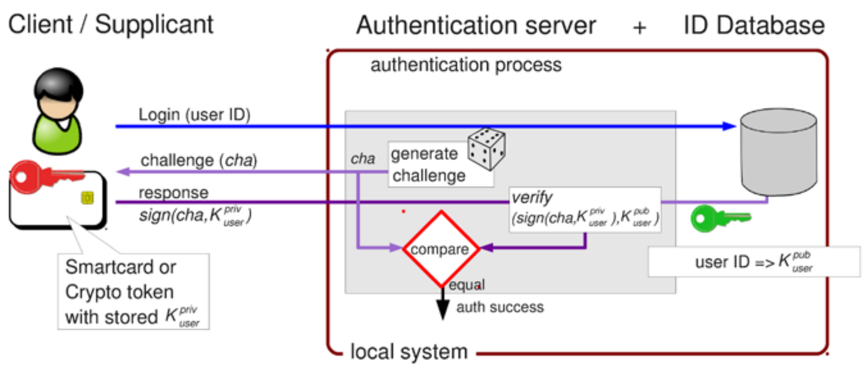
Client / Supplicant
Authentication server + ID Database
authentication process
Login (user ID)
challenge (cha)
response
sign(cha,K^{priv}_{user})
cha
generate challenge
verify (sign(cha,K^{priv}_{user}),K^{pub}_{user})
compare
equal
auth success
user ID => K^{pub}_{user}
Smartcard or Crypto token with stored K^{priv}_{user}
local system

Figure 3. Authentication using signature algorithm with asymmetric keys

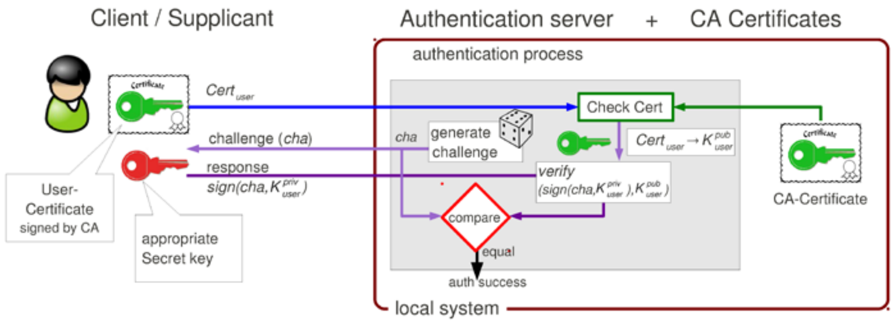

Figure 4. Asymmetric cryptographic authentication with user certificates

response $f(cha,K_{user})$. This leads to authentication schemes based on asymmetric cryptography.

Asymmetric cryptography uses key pairs which can also be used for authentication. Figure 3 demonstrates the operating principle. The client calculates the signature of the generated challenge with her stored private key K_{user}^{priv} and sends the response to the server authentication process. The process verifies the signature by using the public key K_{user}^{pub} of the user, which is stored in the database or which can be derived from the user certificate. The authentication is successful if the decrypted signature is equal to the challenge value. An appropriate asymmetric algorithm, e.g., RSA, ensures that it is hard to recover the key K_{user}^{pub} from the challenge and response.

This method is ownership factor-based, the factor is the private key K_{user}^{priv}. Only the supplicant needs her private key. The authentication service can verify the identity by applying the user's public key K_{user}^{pub} only.

Certificate-based Authentication

The asymmetric authentication mechanism shown in Fig. 3 can be enhanced by certificates. Using certificates, an authentication service must not store the public keys of all its users, but each public key is certified by a trusted party, a certificate authority (CA). This certificate is presented by the client to the authentication service which verifies against the stored CA certificate. If the check holds the public key from the certificate is used for a challenge-response authentication (see Fig. 4).

Thus only the CA certificate has to be stored by the authentication service. The problem is that a public key infrastructure (PKI) has to be established and certificate revocation methods for compromised private user keys have to be implemented. Most operating systems and some user programs, as, e.g., the Mozilla suites, deliver root CA certificates. Thus these clients can connect to servers which use certificate drawn up by these CAs or by sub-CAs. Otherwise the user herself has to decide on the acceptance of the certificate.

The reverse works analogously. If the client has a certificate drawn up by a trustworthy CA, the server may accept the user as authentic. However, the server has to check the ownership of the related secret key (Fig. 4). This can easily be done, as the certificate-based authentication is included in the SSL/TLS standard and is obligatory for server to client authentication. Moreover, the opponent direction from client to server is also a part of the standard. Because SSL/TLS is widely used as secure tunnel protocol for other higher level application protocols, e.g., HTTP, POP3, IMAP, SMTP, and so on, the certificate-based authentication of clients can be easily adapted to the associated applications [10] [11].

The additional administration costs, e.g., for the PKI service, amortize by higher security and fewer costs for helpdesk support, which is needed, e.g., in the user-name/password scenario. No passwords are used which is both, user-friendly as the user does not need keeping passwords in mind and more secure as argued in the introduction. In certificate-based authentication, the same certificate may be used to more than one authentication target as long as suitable asymmetric algorithms are used and the private keys are both, long enough with acceptable quality to prevent brute force attacks and protected against theft.

To raise the grade of security even further, the private key should be stored in non-copyable manner. In doing so, the private key is particularly protected against social engineering attacks. Hardware tokens are suitable for that purpose.

3. HARDWARE TOKEN AUTHENTICATION

Hardware token is the generic term for easy transportable hardware containing microprocessor, volatile and nonvolatile memory, operating system and interfaces for computer connections. Two models are frequently used: smartcards and sticks with USB connectors. The model however is not a vital point. The decisive fact is the operating system and the security-engineered design which allows the secure storage of secret information in non-volatile token memory, not readable from outside the token but only usable by the token processor itself and protected against attacks, e.g., fault attacks and side channel analysis [12].

If the token processor supports symmetric and asymmetric cryptographic operations with stored keys, it is called *cryptographic token* (or *crypto token*, CT). A CT allows the transportable and secure storage of keys. There is no need to read the keys out of the token as all cryptographic

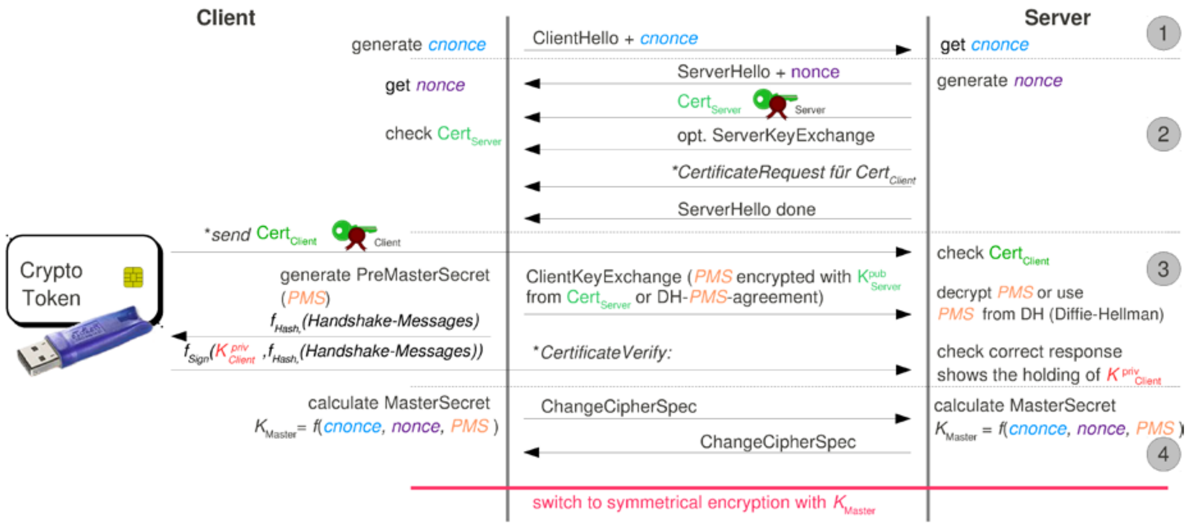
Client
Server
generate cnonce
ClientHello + cnonce
get cnonce
get nonce
ServerHello + nonce
generate nonce
Cert Server
check Cert Server
opt. ServerKeyExchange
*CertificateRequest für Cert Client
ServerHello done
*send Cert Client
Crypto Token
check Cert Client
generate PreMasterSecret (PMS)
ClientKeyExchange (PMS encrypted with K pub Server from Cert Server or DH-PMS-agreement)
decrypt PMS or use PMS from DH (Diffie-Hellman)
f Hash (Handshake-Messages)
f Sign (K priv Client, f Hash (Handshake-Messages))
*CertificateVerify:
check correct response shows the holding of K priv Client
calculate MasterSecret
K Master = f(cnonce, nonce, PMS)
ChangeCipherSpec
switch to symmetrical encryption with K Master
1
2
3
4

Figure 5. Establishing a SSL/TLS connection with CT based authentication

operations are internally performed by the token processor itself. This makes CTs well suitable for authentication purposes.

3.1. Tokens used by certificate-based authentication

As keys and certificates may be stored in hardware tokens and CTs allow cryptographic operations with data stored in the token, CTs are suitable for certificate-based authentication. Moreover, albeit hardware tokens may be stolen, they are protected against misuse by a secret pin (or by even more secure biometric- or key-based authentication to the token).

After the token is unlocked the keys are accessible by the token processor for cryptographic operations. The user connects the token with the computer system, unlocks the keys and let the token response to the challenge request with a digital signature. This may be used for local and remote authentication.

To illustrate the overall process, we look at the establishment of an SSL/TLS connection for building a secure client-server tunnel using client-based authentication. Figure 5 summarizes the process – the protocol steps which are optional for SSL/TLS are denoted by mark *. The client starts the connection to the remote server (1) and sends a *cnonce* (client number used once) value. The server (2) answers with its own *nonce* value and the server certificate. Optionally the server may request a client certificate to inquiry the client authentication. The server certificate is checked by the client (by checking the signature of the issuing authority) and if the check holds, the client sends the *PreMasterSecret PMS* encrypted with the public key from the server certificate in step (3). This is the server part of authentication, because only the server with appropriate secret key may decrypt the *PMS*. Client authentication starts in step (3) by showing the certificate picked from the CT. After the *PMS* generation the client proves its authenticity. It calculates a hash value over all messages sent in the steps before and lets the CT sign this value with the secret key which is associated with the certificate. This value is sent to the server. The server trusts the client if the certificate check holds and the hash value matches and has a valid signature. To perform this test, basically only the public key from the client certificate has to be applied. If both authentications are successful, both, server and client finish the connection establishment. They calculate the master key from the values *nonce*, *cnonce* and *PMS* and switch to a symmetric encryption method for data exchange in the established tunnel.

3.2. Client token access

To use tokens in client-based authentication a connection between the client and the token is required. There are several hard- and middleware solutions, but only a few are token hardware and operating system independent and allow the use of several crypto tokens.

One solution is PKCS#11 (public-key cryptography standards no. 11 published by RSA Security [13]). PKCS#11 defines an API (Cryptoki) for a generic interface to cryptographic tokens. This API allows the generation of keys, the storage of certificates and the usage of several cryptographic methods within the token. The standard is token vendor independent and well supported. It is used in form of a dynamic library, which can be bound to several programs to access the token as a cryptographic device with a unique API.

In particular, the applications from the Mozilla suite support PKCS#11. These applications cover the essential user requirements for network access: they allow to surf the web and to send or to receive emails. For SSL/TLS capable servers the certificate-based authentication with crypto token by using PKCS#11 is possible, too.

The drawback of the approach discussed is that certificate-based approaches require a public key infrastructure (PKI) and adequate transport methods for the associated keys and, thus, unfortunately increase the efforts not only for service providers but also for customers. To raise the users' acceptance, new capabilities and easier authentication mechanisms have to be adopted. This leads us to combine the certificate-based authentication mechanism using cryptographic hardware tokens with single sign-on (SSO) software. Moreover, the certificates and the associated keys enable additional usage capabilities like electronic signatures or encipherment. This should help to make the approach of certificate-based authentication attractive to users.

4. SSO AS ADDITIONAL FUNCTIONALITY

As already mentioned in the previous section, one of the mostly used standards for token access is the PKCS#11 specification with Cryptoki as API. This software interface hides hardware details to a large extent.

To access private objects stored on an hardware token, applications which use Cryptoki have to be firstly authenticated to the token by asking the user for a pin. Thus the user has to put in her pin every time an application is called. This may be a nuisance. To avoid multiple inputs of a pin by one and the same user, each user should be encapsulated in a user session during the login to the operating system. That way separated from other users, each user has to put in her pin only once during a session. After acceptance of the pin, the hardware token could be used by every application of the user session in question without calling the user back. In the following, we discuss in detail such a token-based SSO approach.

There already exist vendor specific multi-application SSO solutions in combination with hardware tokens which use the tokens as password safes. Our approach targets SSO as combination of vendor independent hardware token and the more secure certificate-based user authentication instead of password authentication [14].

Let us review in detail how Cryptoki is commonly used by an application. The process consists of 4 steps:

a) The PKCS#11 library is opened and tokens are searched for by scanning the appropriate slots.

b) An application session to the token is started which allows the application for reading the public objects of the token.

c) The user has to be authenticated to the token before the application may use private objects.

d) After usage the logout procedure is called and the application session and the libraries are closed.

To adapt the approach to SSO, steps (c) and (d) are the interesting one. In step (c), the PKCS#11 login procedure `C_Login()` asks the user for authenticating herself, e.g., by input of the pin. The login status persists until the `C_Logout()` procedure is called in step (d). However, even during this time period only the corresponding application session is permitted to use private objects on the token. Each other opened application session requires an extra

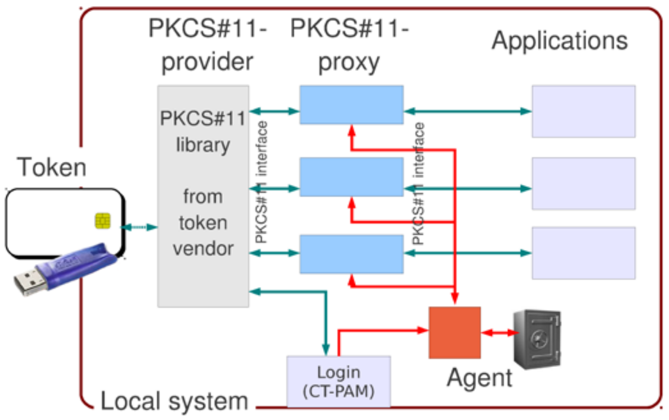

Figure 6. PKCS#11 extension for SSO

authentication of the user, even if the application is called by one and the same user. From the perspective of the user the required authentication steps could be reduced, if the pin is temporary stored during the first authentication step and inserted automatically for following logins of applications running in the same operating system user session. The pin has to be stored until the last application logs out from the token or the user logs out from the operating system.

A way to arrive at SSO is to use an agent which stores the pin (not the passwords like other agents do) and which is asked when a process needs to authenticate to the token. Figure 6 shows our approach. We aimed at not having to modify the applications themselves. We only extend the PKCS#11 interface in the local machine.

A transparent layer between the applications and the PKCS#11 interface is introduced. This intermediate layer plays the role of a proxy which connects the applications to the PKCS#11 interface and works in conjunction with an agent which stores the user's pin. The agent functionality is known from the widely used SSH-agent [15], but in contrast to it, no decrypted secret key is stored in the agent but only the pin.

The new layer works as follows. First, a user logs into the operating system by authenticating herself. For this purpose, she connects her crypto token with the interface and – in the login screen – enters her pin. The *CT-PAM* (PAM: Pluggable Authentication Modules) gets the pin and logs into the token, verifies the certificate, and starts a challenge to the private key in the token. If the response is correct, the login is successful. During the login process, the pin passed to the vendor PKCS#11 library is duplicated and stored into the agent. In further steps, the applications do not use the vendor specific PKCS#11 library directly. They connect to the intermediate PKCS#11 library which offers the same interface as described in [13]. The intermediate library is denoted as PKCS#11 proxy in Fig. 6. Most interface calls are delegated from the intermediate to the vendor library without modifications. When an application has to authenticate to the token, the agent is asked for the pin.

For this purpose, the intermediate PKCS#11 library tells all applications with a special flag (CKF_PROTECTED_-AUTHENTICATION_PATH) that there is a ”protected authentication path”, which means that a user can log in to the token without passing a pin. As a consequence the application does not need to ask for a pin.

We have successfully implemented a system which provides the functionality of the proposed approach [14]. For this purpose we have extended the PAM PKCS#11 library so that the agent is started after a successful login with a given pin. Among others, *Firefox*, *Thunderbird* and *OpenSSH* have been used as user applications in the test system which works out-of-the-box.

In conjunction with our SSO solution we have to ensure that the pin stored in the system’s main memory cannot be used by unauthorized applications. To get this point under control, the system has to check whether the source of the connection is an authorized application when a connection to the interface library is done. Authorized applications could be defined by an administrator using software signatures or checksums as footprint. The agent could check the footprint and ensure that only defined applications are accepted.

Another important item directly concerns the storage of the pin in the system’s main memory. This location has to be protected against malicious attacker programs. To ensure that a scan of the whole main memory cannot reveal the pin in clear form, the agent generates a key, encrypts the pin with this key and decrypts it when necessary. By this, it is rather difficult to get knowledge of the pin as the key has to be located and extracted, too.

The agent detects the removal of the hardware token. When the token is unplugged, the pin is deleted from the system’s main memory whereby the memory addresses are overwritten with random values.

Securing the system as described should lead to an SSO environment without (much more) higher security risk than the risk of a system in which the pin is asked for every time a Cryptoki session is started.

5. CONCLUSION

Certificate-based authentication with safe transport of secret keys may reduce the risks in authentication processes if used as comprehensive replacement for passwords or other secret knowledge-based mechanisms. In view of the better security, technically interested persons have no problems at all to switch from passwords to authentication with smartcards or USB crypto tokens. However, as the approach requires entrainment of additional hardware, the populace do not accept the new technology if raise of security is the only argument. In order to raise the users’ acceptance the approach has to be expanded by further functionalities which ease the users’ tasks without losing of security.

In this paper, we have proposed to enrich the approach by single sign-on, electronic signatures, and encipherment; a single sign-on implementation has been presented in detail. With these additional features in hand, we could convince the rectorate of our university (with more than 18,000 students) of stepwise introducing token-based authentication.

6. ACKNOWLEDGEMENTS

This work has been supported by the European Commission, European Funds for Regional Development, Saxony- Anhalt 2007-2013.

REFERENCES

[1] S. U. Shah, Fazl-e-Hadi, and A. A. Minhas. New factor of authentication: something you process. In: Proceedings of the international conference on future computer and communication (ICFCC), pp. 102–106, IEEE, (2009) Apr 3–5, Kuala Lumpar, Malaysia.

[2] J. Brainard, A. Juels, R. L. Rivest, M. Szydlo, and M. Yung. Fourth-factor authentication: somebody you know. In: Proceedings of the 13th conference on computer and communications security (CCS), pp. 168–178, ACM, (2006) Oct 30 – Nov 3, Alexandra, Virginia, USA.

[3] D. Florencio and C. Herley. A large-scale study of web password habits. In: Proceedings of the 16th international world wide web conference (WWW), pp. 657–666, ACM, (2007) May 8–12, Banff, Alberta, Canada.

[4] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: user attitudes and behaviors. In: Proceedings of the 6th symposium on usable privacy and security (SOUPS), pp. 2:1–2:20, ACM, (2010) July 14–16, Redmond, Washington, USA.

[5] C. Herley. So long, and no thanks for the externalities: the rational rejection of security advice by users. In: Proceedings of the 2009 new security paradigms workshop (NSPW), pp. 133–144, ACM, (2009) Sep 8–11, Oxford, United Kingdom.

[6] A. Adams and M. A. Sasse. Users are not the enemy. In: Communications of the ACM, 42(12):40–46, ACM, (1999).

[7] A. P. Pons and P. Polak. Understanding user perspectives on biometric technology. In: Communications of the ACM, 51(9):115– 118, ACM, (2008).

[8] F. AL-Harby, R. Qahwaji, and M. Kamala. Users’ acceptance of secure biometrics authentication system: reliability and validate of an extended UTAUT model. In: Networked digital technologies, ser. Communications in computer and information science, vol. 87, pp. 254–258, Springer, (2010).

[9] T. Weigold, T. Kramp, and M. Baentsch. Remote client authentication. In: IEEE Security and Privacy, 6(4):36–43, IEEE, (2008).

[10] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol – version 3.0. Transport layer security working group, Internet draft, (1996).

[11] C. Newman. Using TLS with IMAP, POP3 and ACAP. Internet engineering task force, RFC 2595, (1999).

[12] W. Schindler, K. Lemke, and C. Paar. A stochastic model for differential side channel cryptanalysis. In: Proceedings of the 7th

international conference on cryptographic hardware and embedded systems (CHES), pp. 30–46, Lecture Notes in Computer Science, LNCS 3659. Springer, (2005).

[13] RSA Laboratories. PKCS #11: Cryptographic Token Interface Standard. http://www.rsa.com/rsalabs/node.asp?id=2133, 2004. [Online]. Available: http://www.rsa.com/rsalabs/node.asp?id=2133

[14] S. Wefel and P. Molitor. Client hardware-token based single sign-on over several servers without trusted online third party server. In: Proceedings of the 3rd international conference on security and assurance (ISA), Advances in information security and its spplication, pp. 29–36, Springer, ser. Communications in computer and information science (CCIS), vol. 36, (2009) June 25–27, Seoul, Korea.

[15] D. J. Barrett, R. E. Silverman, and R. G. Byrnes. SSH, The Secure Shell: The Definitive Guide (2nd ed). O'Reilly Media, (2005).